

# Pose Tracking I

Gordon Wetzstein  
Stanford University

EE 267 Virtual Reality

Lecture 11

[stanford.edu/class/ee267/](http://stanford.edu/class/ee267/)



# Overview

- overview of positional tracking
- camera-based tracking
- HTC's Lighthouse
- VRduino – an Arduino for VR, specifically designed for EE 267 by Keenan Molner
- pose tracking with VRduino using homographies

# What are we tracking?

- Goal: track pose of headset, controller, ...
- What is a pose?
  - 3D position of the tracked object
  - 3D orientation of the tracked object, e.g. using quaternions or Euler angles
- Why? So we can map the movement of our head to the motion of the camera in a virtual environment – motion parallax!

# Overview of Positional Tracking

“inside-out tracking”: camera or sensor is located on HMD, no need for other external devices to do tracking

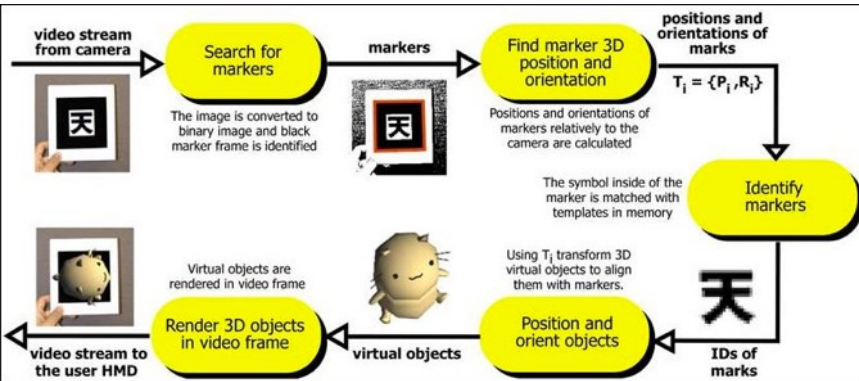
- simultaneous localization and mapping (SLAM) – classic computer & robotic vision problem (beyond this class)

“outside-in tracking”: external sensors, cameras, or markers are required (i.e. tracking constrained to specific area)

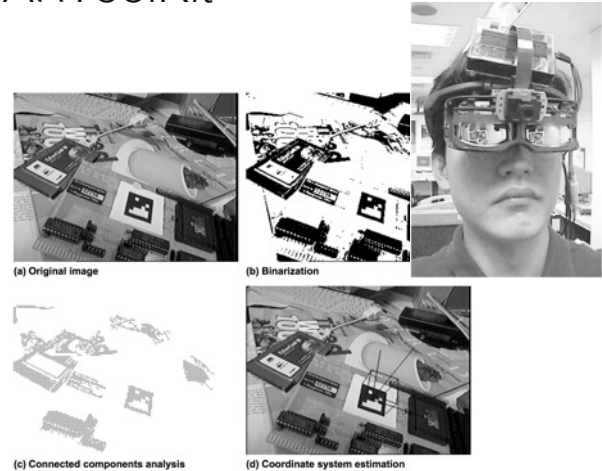
- used by most VR headsets right now, but everyone is feverishly working on insight-out tracking!

# Marker-based Tracking

- seminal papers by Rekimoto 1998 and Kato & Billinghurst 1999
- widely adopted after introduced by ARToolKit

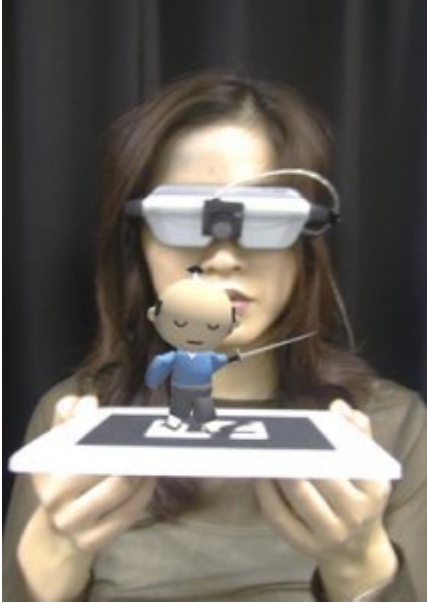


Kato, Billinghurst - ARToolKit



Rekimoto - Matrix

# Marker-based Tracking



ARToolKit

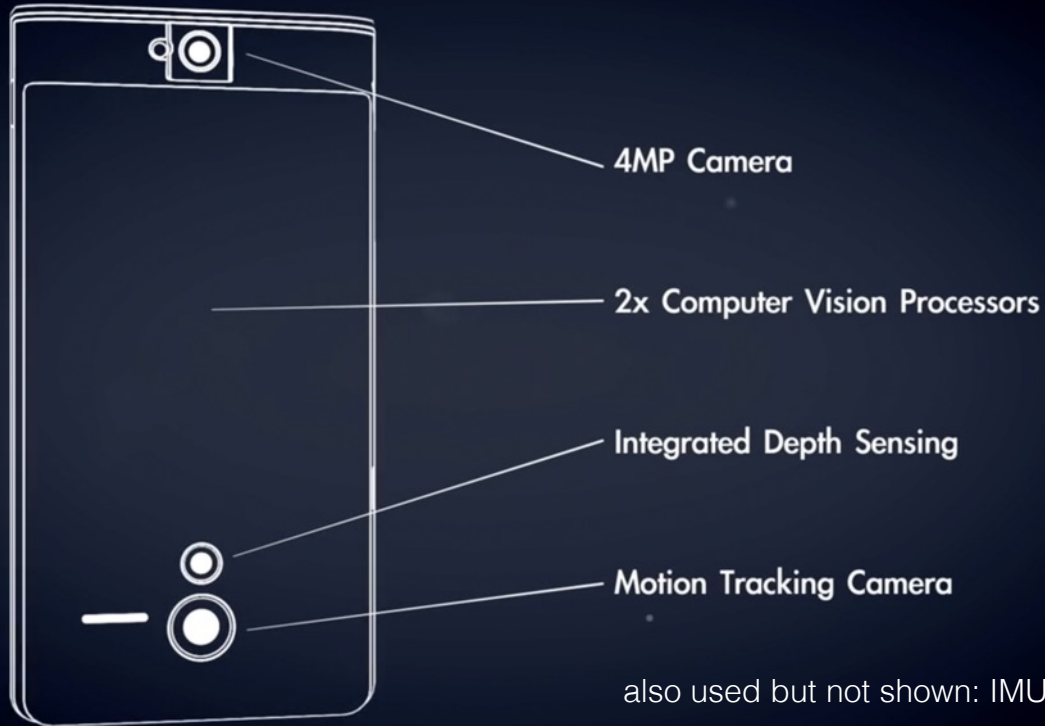


OpenCV marker tracking

# Inside-out Tracking

Google's Project Tango





also used but not shown: IMU

problem: SLAM via sensor fusion



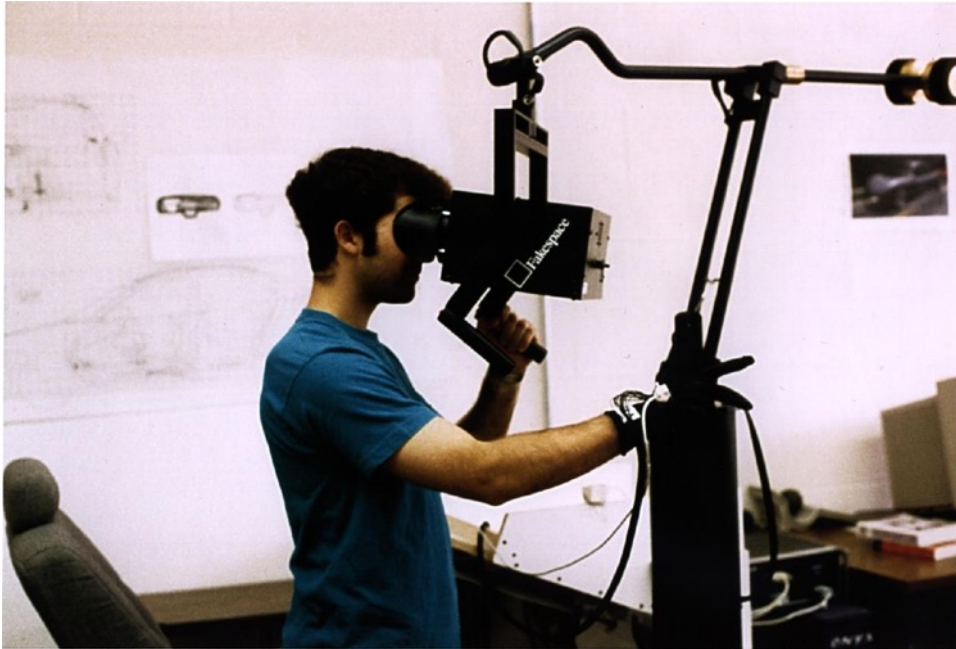
# Inside-out Tracking

- marker-less inside-out tracking used by Microsoft HoloLens, Oculus Quest, Magic Leap, ...
- eventually required by all untethered VR/AR systems
- if you need it for your own HMD, consider using Intel's RealSense (small & has SDK)
- if you want to learn more about SLAM, take a 3D computer vision or robotic vision class, e.g. Stanford CS231A

# “Outside-in Tracking”

- mechanical tracking
- ultra-sonic tracking
- magnetic tracking
- optical tracking
- GPS
- WIFI positioning
- marker tracking
- ...

# Positional Tracking - Mechanical



some mechanical linkage, e.g.

- fakespace BOOM
- microscribe



# Positional Tracking - Mechanical

pros:

- super low latency
- very accurate

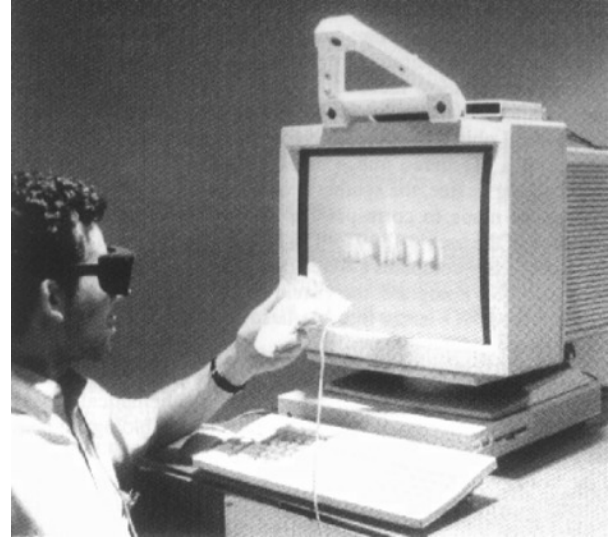
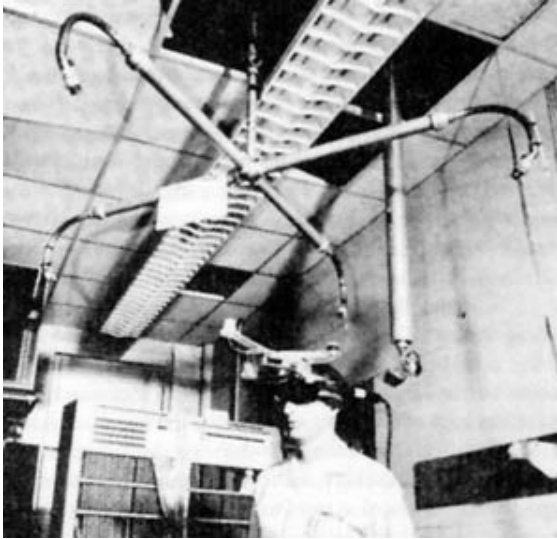
cons:

- cumbersome
- “wired” by design

# Positional Tracking – Ultra-sonic

- 1 transmitter, 3 receivers  $\rightarrow$  triangulation

Ivan Sutherland's "Ultimate Display"



Logitech 6DOF

# Positional Tracking – Ultra-sonic

pros:

- can be light, small, inexpensive

cons:

- line-of-sight constraints
- susceptible to acoustic interference
- low update rates

# Positional Tracking - Magnetic

- reasonably good accuracy
- position and orientation
- 3 axis magnetometer in sensors
- need magnetic field generator (AC, DC, ...), e.g. Helmholtz coil
- magnetic field has to oscillate and be sync'ed with magnetometers



3 axis Helmholtz coil  
[www.directvacuum.com](http://www.directvacuum.com)

# Positional Tracking - Magnetic

pros:

- small, low cost, low latency sensors
- no line-of-sight constraints

cons:

- somewhat small working volume
- susceptible to distortions of magnetic field
- not sure how easy it is to do this untethered (need to sync)



3 axis Helmholtz coil  
[www.directvacuum.com](http://www.directvacuum.com)



# Positional Tracking - Magnetic



Magic Leap One controller tracking:

- magnetic field generator in controller
- magnetometer in headset



# Positional Tracking - Optical

- track active (near IR) LEDs with cameras →

OR

- track passive retro-reflectors with IR illumination around camera
- both Oculus Rift and HTC Vive come with optical tracking



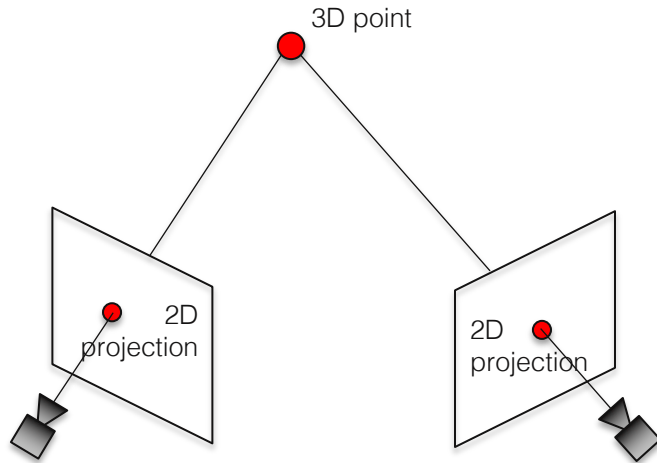
Oculus Rift

<https://www.ifixit.com/Teardown/Oculus+Rift+C V1+Teardown/60612>



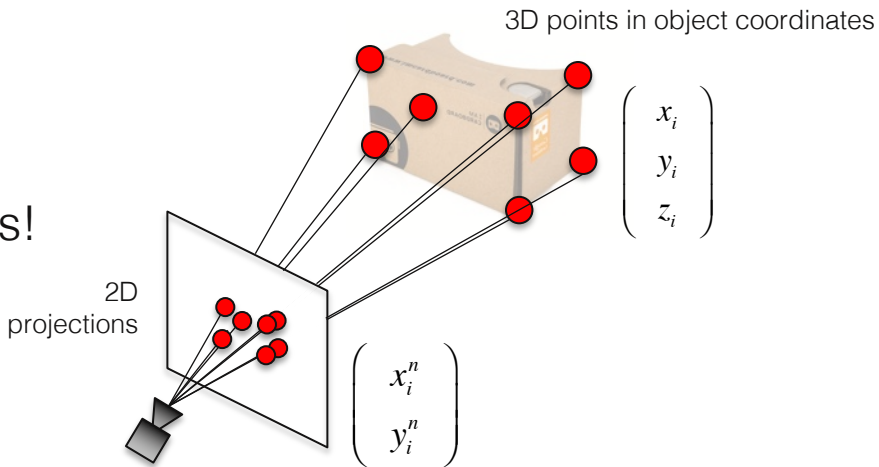
# Understanding Pose Estimation - Triangulation

- for tracking individual 3D points, multi-camera setups usually use triangulation
- this does not give us the pose (rotation & translation) of camera or object yet



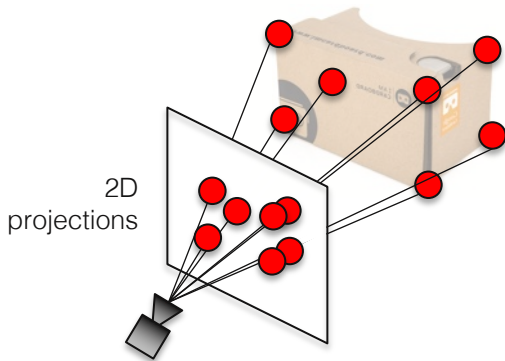
# Understanding Pose Estimation

- for pose estimation, need to track multiple points with known relative 3D coordinates!



# Understanding Pose Estimation

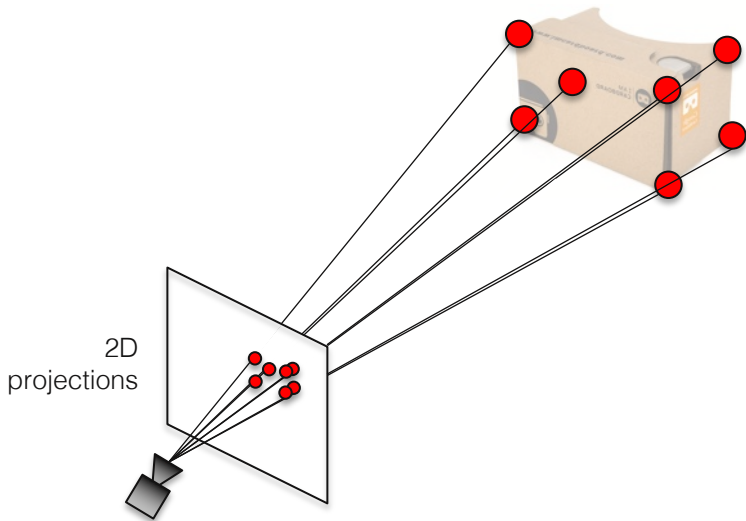
- when object is closer, projection is bigger



# Understanding Pose Estimation

- when object is farther,  
projection is smaller

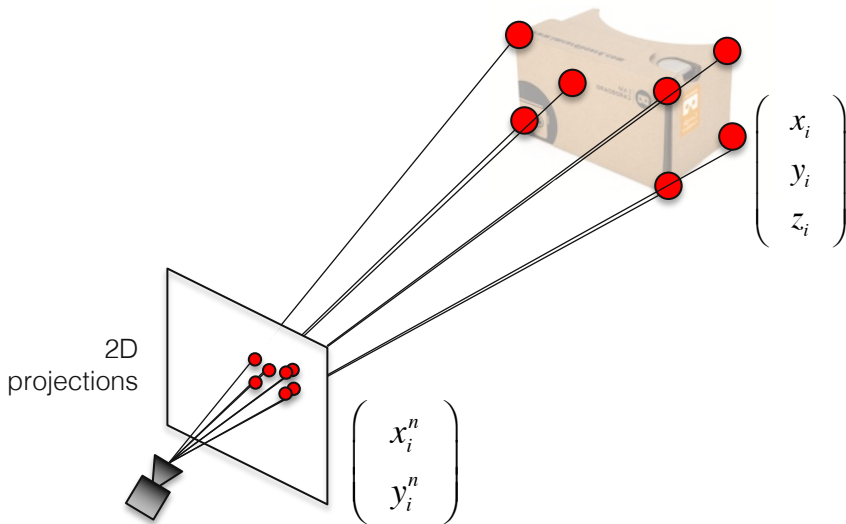
... and so on ...



Estimating 6-DoF pose from 2D projections is known as the  
*Perspective-n-point problem!*

# Understanding Pose Estimation

1. how to get projected 2D coordinates?
2. image formation
3. estimate pose with linear homography method
4. estimate pose with nonlinear Levenberg-Marquardt method (next class)





# Understanding Pose Estimation

1. how to get projected 2D coordinates?
  - HTC Lighthouse
  - VRduino
2. image formation
3. estimate pose with linear homography method
4. estimate pose with nonlinear Levenberg-Marquardt method (next class)

# HTC Lighthouse

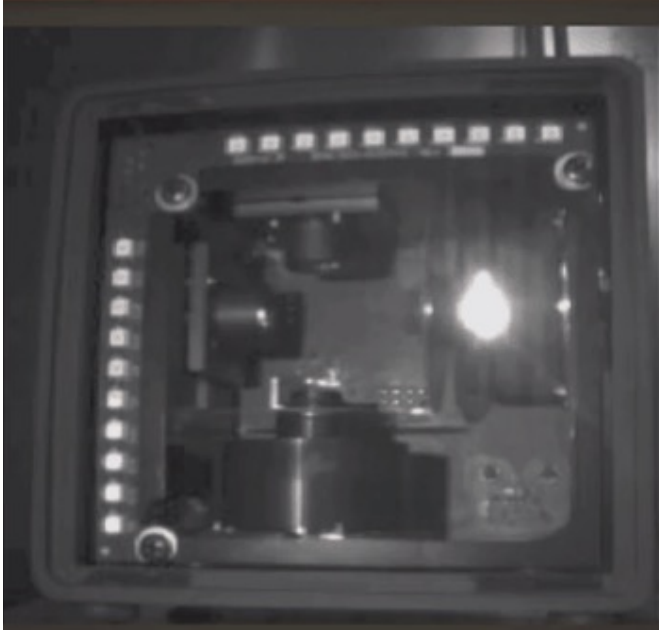


# HTC Lighthouse



<https://www.youtube.com/watch?v=J54dotTt7k0>

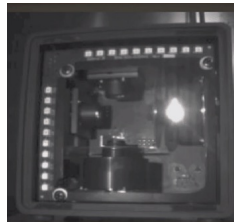
# HTC Lighthouse – Base Station



# HTC Lighthouse – Base Station

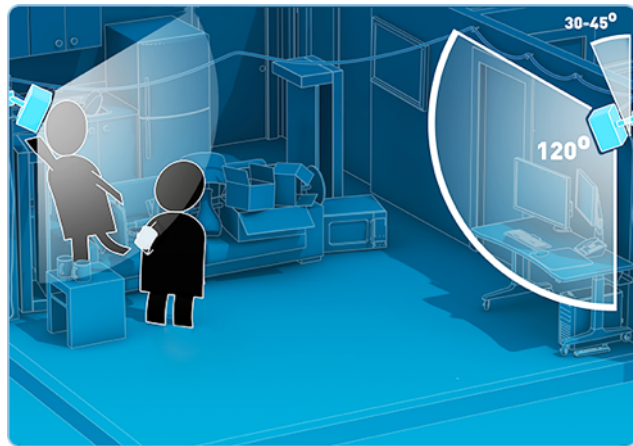
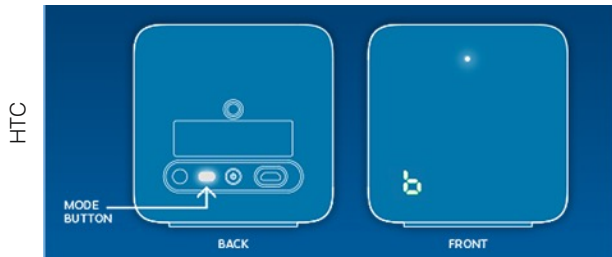
important specs:

- runs at 60 Hz
  - i.e. horizontal & vertical update combined 60 Hz
  - broadband sync pulses in between each laser sweep (i.e. at 120 Hz)
- each laser rotates at 60 Hz, but offset in time
- useable field of view: 120 degrees



# HTC Lighthouse – Base Station

- can use up to 2 base stations simultaneously via *time-division multiplexing* (TDM)
- base station modes:
  - A: TDM slave with cable sync
  - B: TDM master
  - C: TDM slave with optical sync



# HTC Lighthouse – Base Station

- sync pulse periodically emitted (120 times per second)
- each sync pulse indicates beginning of new sweep
- length of pulse also encodes additional 3 bits of information:

Name	skip	data	axis	length (ticks)	length (μs)
j0	0	0	0	3000	62.5
k0	0	0	1	3500	72.9
j1	0	1	0	4000	83.3
k1	0	1	1	4500	93.8
j2	1	0	0	5000	104
k2	1	0	1	5500	115
j3	1	1	0	6000	125
k3	1	1	1	6500	135

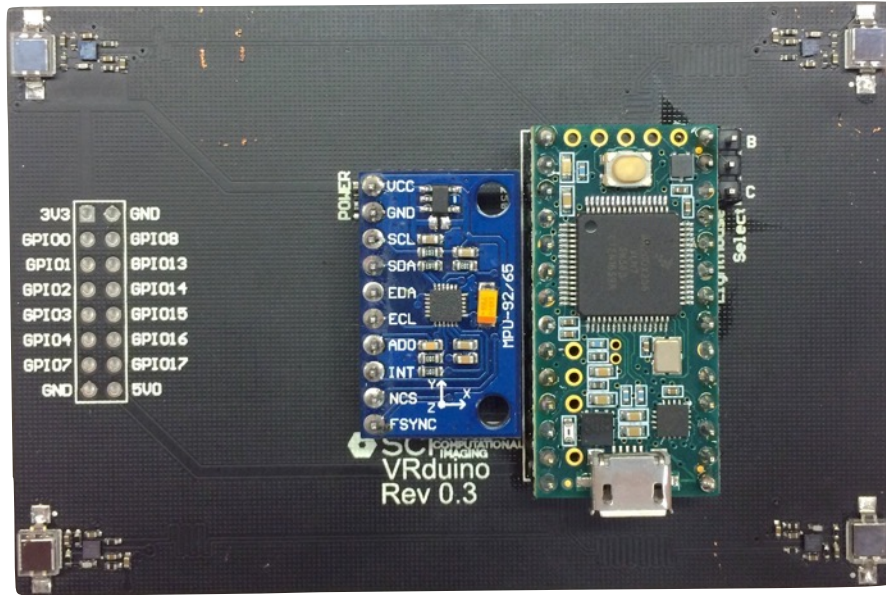
- axis: horizontal or vertical sweep to follow
- skip: if 1, then laser is off for following sweep
- data: data bits of consecutive pulses yield OOTX frame

# VRduino

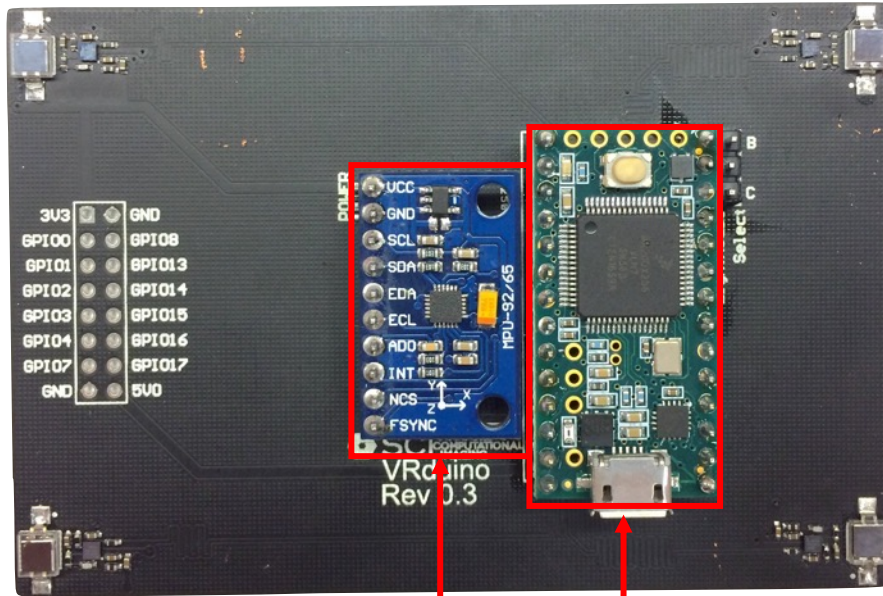
- in this class, we use the HTC Lighthouse base stations but implement positional tracking (i.e., pose estimation) on the VRduino
- VRduino is a shield (hardware add-on) for the Arduino Teensy 3.2; custom-designed for EE 267 by Keenan Molner



# VRduino



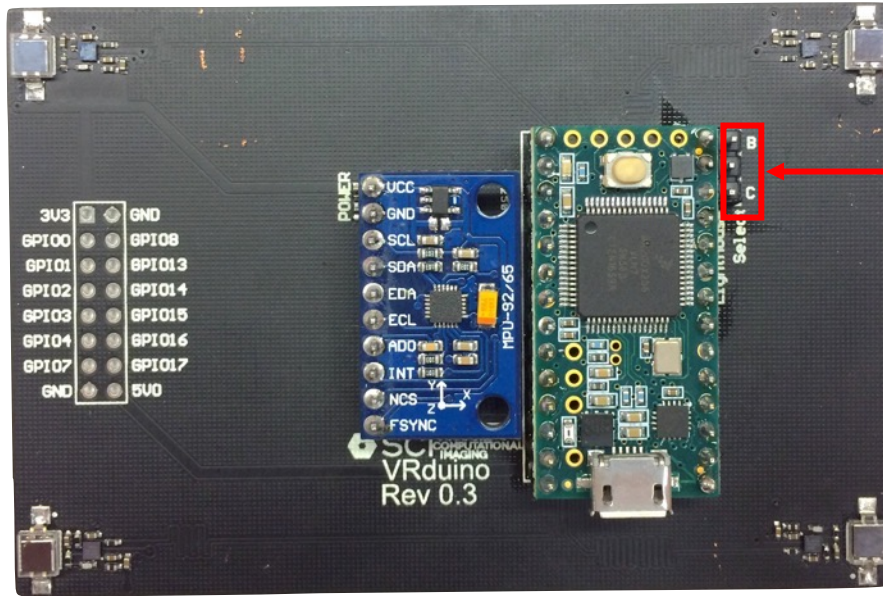
# VRduino



IMU

Teensy 3.2

# VRduino



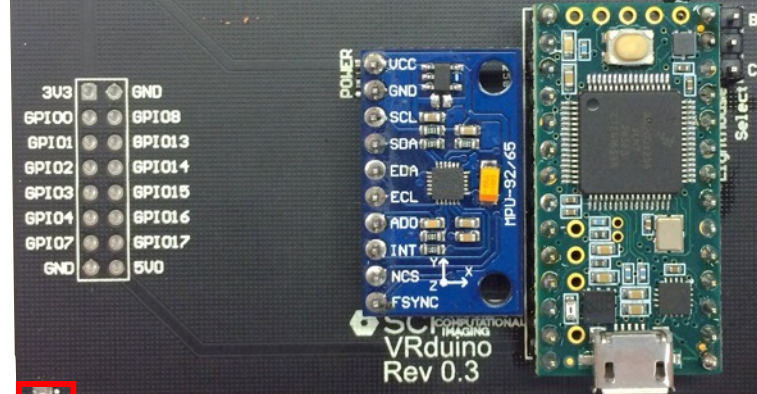
Lighthouse  
Select

# VRduino

Photodiode 0



Photodiode 1



Photodiode 3



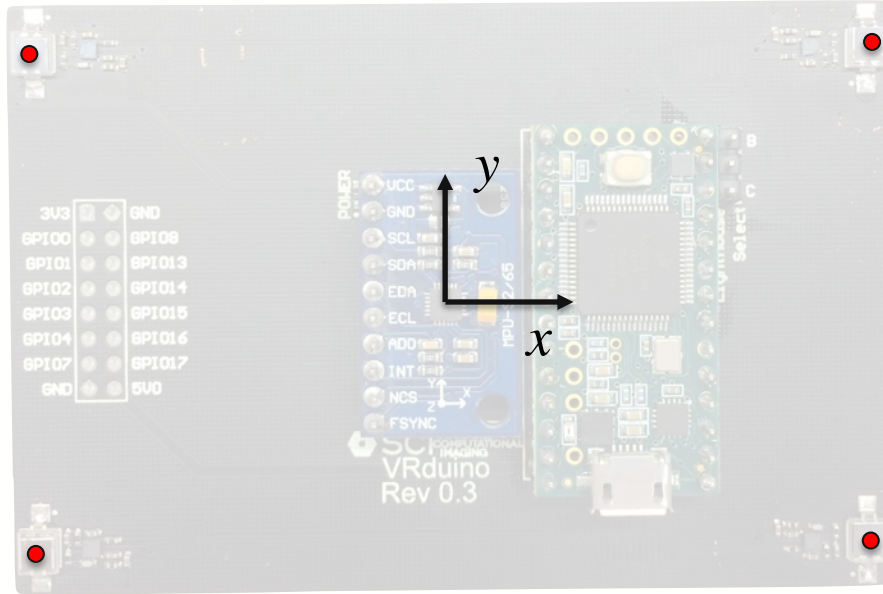
Photodiode 2



# VRduino

$x=-42\text{mm}$ ,  $y=25\text{mm}$

$x=42\text{mm}$ ,  $y=25\text{mm}$

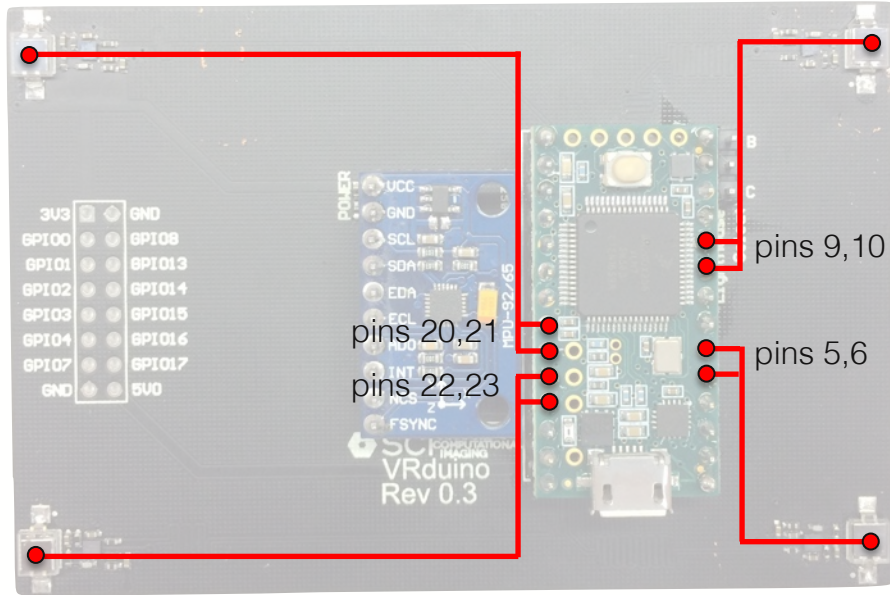


$x=-42\text{mm}$ ,  $y=-25\text{mm}$

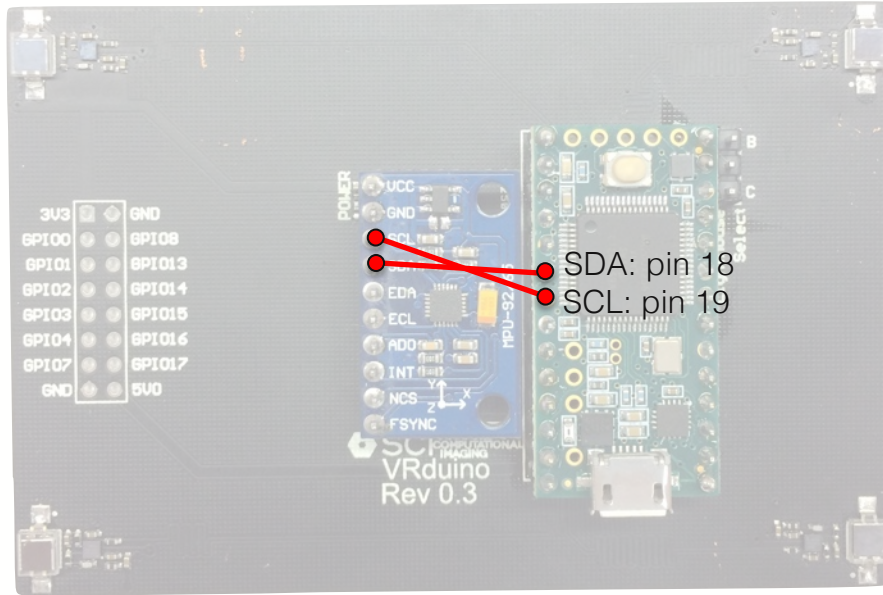
$x=42\text{mm}$ ,  $y=-25\text{mm}$



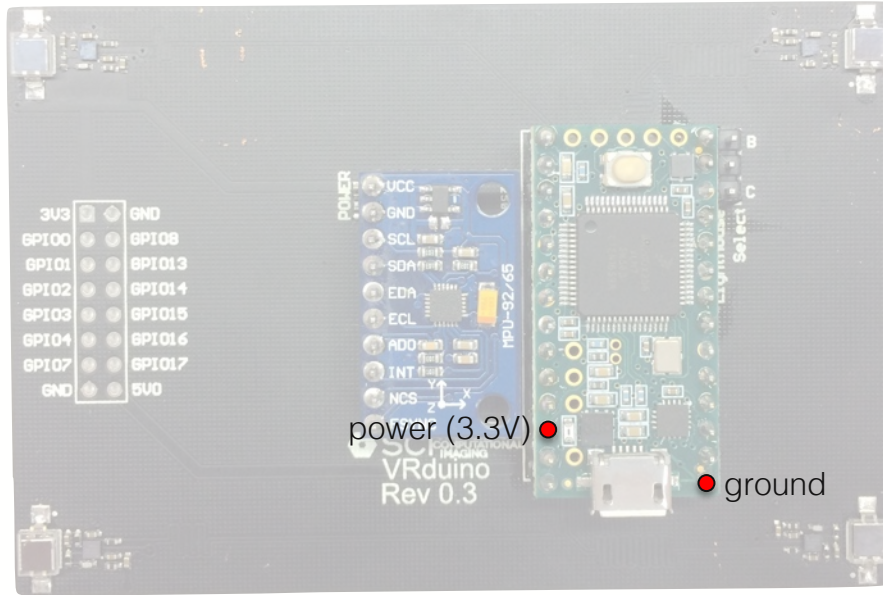
# VRduino



# VRduino

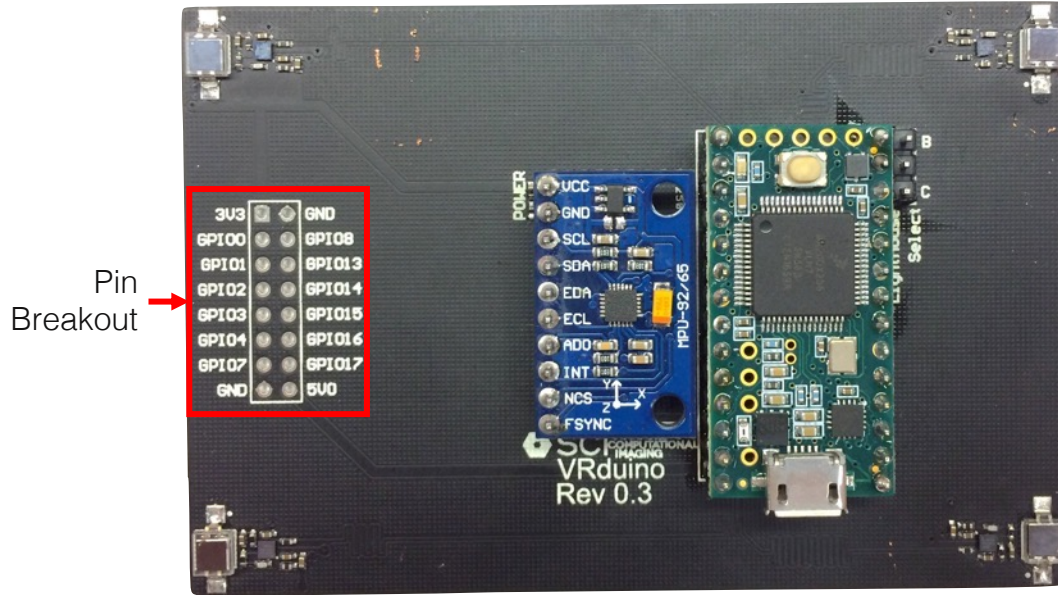


# VRduino



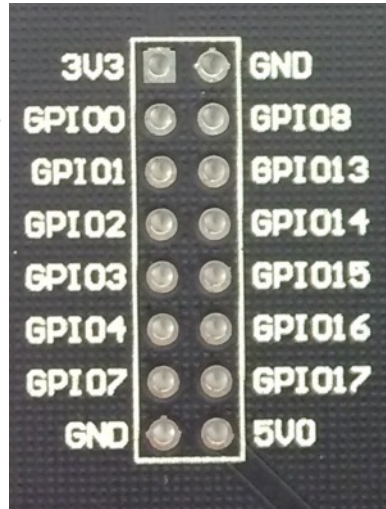


# VRduino



# VRduino

3.3V power, **200mA MAX**  
digital R/W, Serial, cap. sense  
digital R/W, Serial, cap. sense  
digital R/W  
digital R/W, PWM, CAN  
digital R/W, PWM, CAN  
digital R/W, Serial, SPI



SPI, Serial, digital R/W  
digital R/W  
SPI, analog read, digital R/W  
SPI, analog read, digital R/W  
I2C, analog read, digital R/W  
I2C, analog read, digital R/W  
5V power, **500mA MAX**

For more details, see Lab Writeup

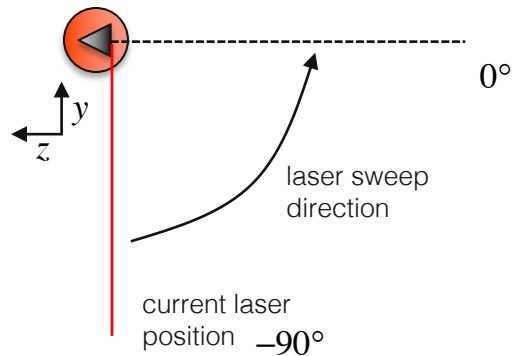
# Pose Estimation with the VRduino

- timing of photodiodes reported in Teensy “clock ticks” relative to last sync pulse
- Teensy usually runs at 48 MHz, so 48,000,000 clock ticks per second

# How to Get the 2D Coordinates?

- at time of respective sync pulse, laser is at  $90^\circ$  horizontally and  $-90^\circ$  vertically
- each laser rotates  $360^\circ$  in  $1/60$  sec

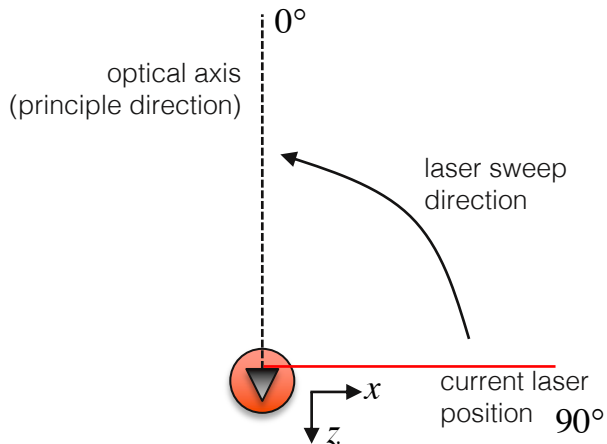
Side View



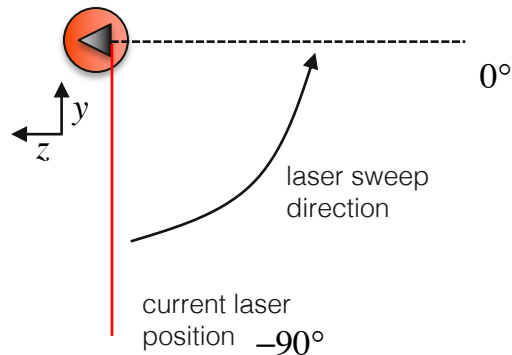
# How to Get the 2D Coordinates?

- at time of respective sync pulse, laser is at  $90^\circ$  horizontally and  $-90^\circ$  vertically
- each laser rotates  $360^\circ$  in  $1/60$  sec

Top View



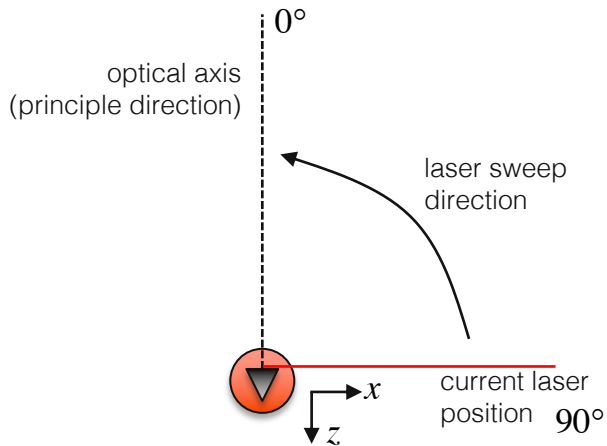
Side View



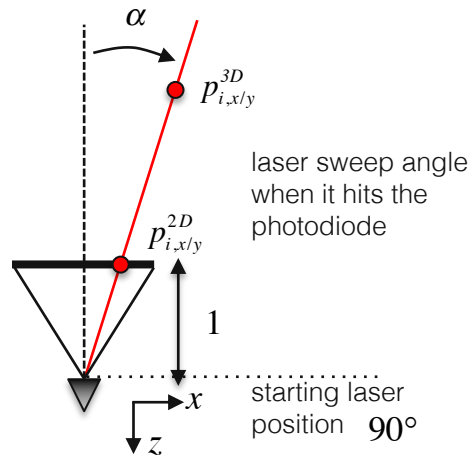
# How to Get the 2D Coordinates?

- at time of respective sync pulse, laser is at  $90^\circ$  horizontally and  $-90^\circ$  vertically
- each laser rotates  $360^\circ$  in  $1/60$  sec
- convert from ticks to angle first and then to relative position on plane at unit distance

Top View



Top View



# How to Get the 2D Coordinates?

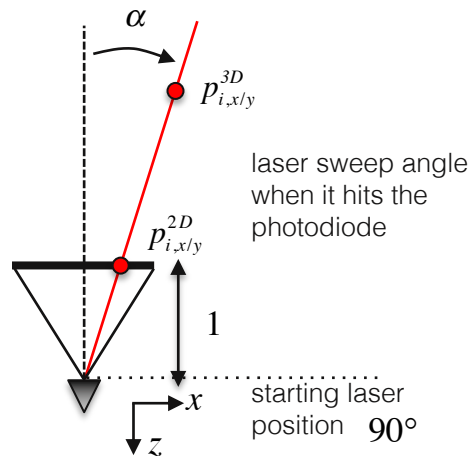
raw number of ticks from photodiode



$$\Delta t [\text{sec}] = \frac{\# \text{ ticks}}{48,000,000 \left[ \frac{\text{ticks}}{\text{sec}} \right]} \leftarrow \text{CPU speed}$$

- convert from ticks to angle first and then to relative position on plane at unit distance

Top View



# How to Get the 2D Coordinates?

raw number of ticks from photodiode



$$\Delta t [\text{sec}] = \frac{\# \text{ ticks}}{48,000,000 \left[ \frac{\text{ticks}}{\text{sec}} \right]} \leftarrow \text{CPU speed}$$

offset from sync pulse

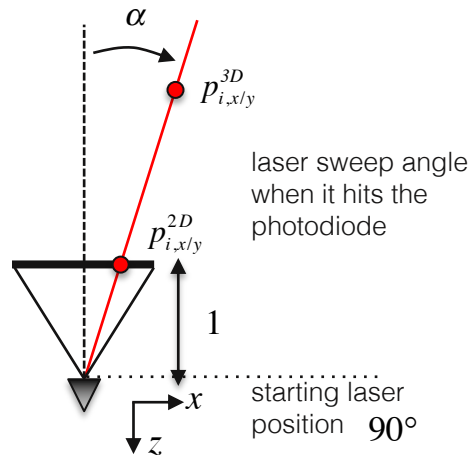


$$\alpha [^\circ] = -\frac{\Delta t [\text{sec}]}{\frac{1}{60} \left[ \frac{\text{sec}}{360} \right]} + \frac{360}{4} [^\circ]$$

time per 1 revolution

- convert from ticks to angle first and then to relative position on plane at unit distance

Top View





# How to Get the 2D Coordinates?

- convert from ticks to angle first and then to relative position on plane at unit distance

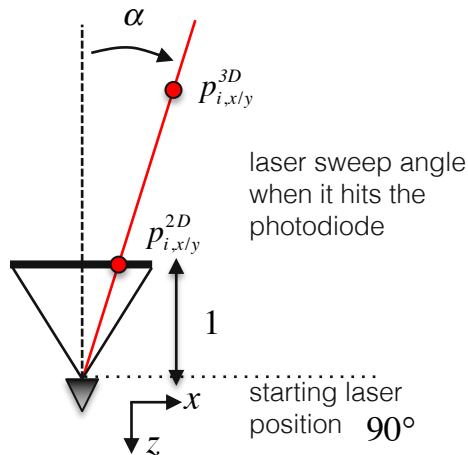
$$p_{i,x/y}^{2D} = \tan\left(\frac{\alpha}{360[^\circ]} \cdot 2\pi\right)$$

offset from sync pulse

$$\alpha[^\circ] = -\frac{\Delta t[\text{sec}]}{\frac{1}{60} \left[ \frac{\text{sec}}{360[^\circ]} \right]} + \frac{360}{4}[^\circ]$$

time per 1 revolution

Top View



# How to Get the 2D Coordinates?

Horizontal Sweep

$$\alpha[^\circ] = -\frac{\Delta t[\text{sec}]}{\frac{1/60}{360}\left[\frac{\text{sec}}{^\circ}\right]} + \frac{360}{4}[^\circ]$$

Vertical Sweep

$$\alpha[^\circ] = \frac{\Delta t[\text{sec}]}{\frac{1/60}{360}\left[\frac{\text{sec}}{^\circ}\right]} - \frac{360}{4}[^\circ]$$

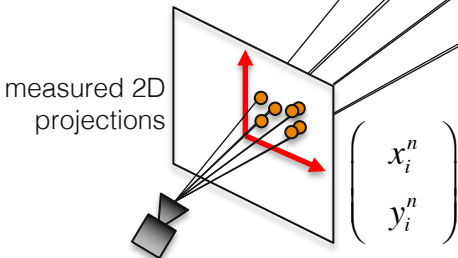
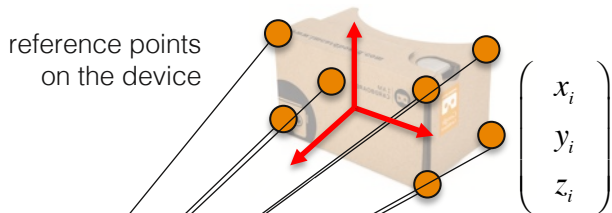
# Understanding Pose Estimation

1. how to get projected 2D coordinates?
  2. image formation
  3. estimate pose with linear homography method
  4. estimate pose with nonlinear Levenberg-Marquardt method (next class)
- how 3D points project into 2D coordinates in a camera (or a Lighthouse base station)
  - very similar to graphics pipeline

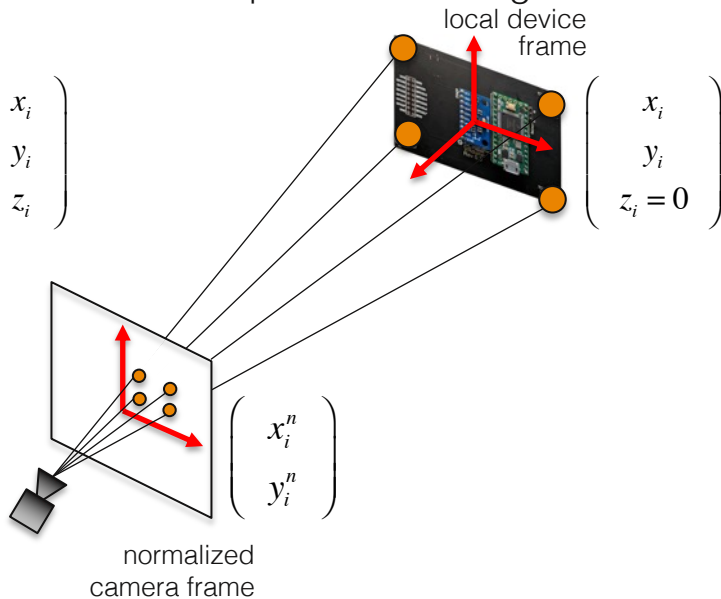
# Image Formation

- image formation is model for mapping 3D points in local “object” coordinate system to 2D points in “window” coordinates

3D reference point arrangement



planar 2D arrangement



# Image Formation – 3D Arrangement

1. transform 3D point into view space:

$$\begin{pmatrix} x_i^c \\ y_i^c \\ \textcircled{w_i^c} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix}$$

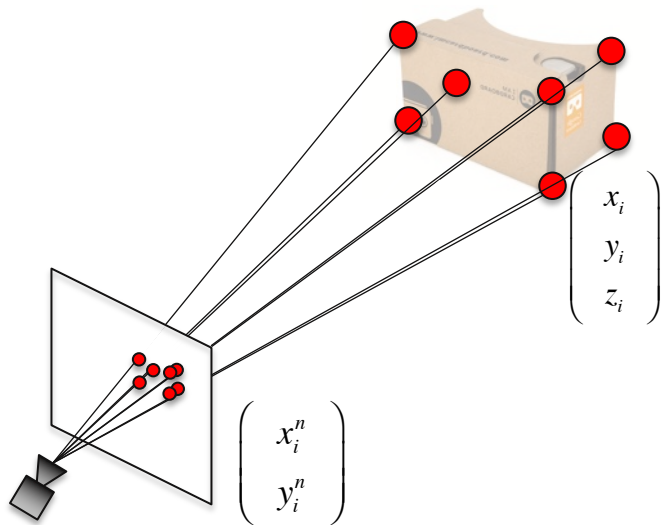
↑  
“projection matrix”

↑  
“modelview matrix”  
3x3 rotation matrix and  
translation 3x1 vector

↑  
This is the homogeneous  
coordinate

2. perspective divide:

$$\begin{pmatrix} x_i^n \\ y_i^n \end{pmatrix} = \begin{pmatrix} \frac{x_i^c}{w_i^c} \\ \frac{y_i^c}{w_i^c} \end{pmatrix}$$



# Image Formation – 2D Arrangement

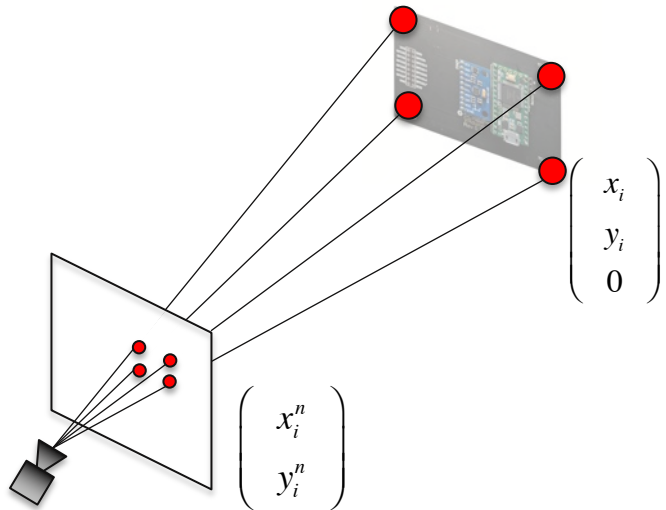
1. transform 3D point into view space:

$$\begin{pmatrix} x_i^c \\ y_i^c \\ w_i^c \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \\ 0 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

2. perspective divide:

$$\begin{pmatrix} x_i^n \\ y_i^n \end{pmatrix} = \begin{pmatrix} \frac{x_i^c}{w_i^c} \\ \frac{y_i^c}{w_i^c} \end{pmatrix}$$



# Image Formation – 2D Arrangement

- all rotation matrices are orthonormal, i.e.  $\sqrt{r_{11}^2 + r_{21}^2 + r_{31}^2} = 1$   
 $\sqrt{r_{12}^2 + r_{22}^2 + r_{32}^2} = 1$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \begin{matrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ r_{31} & r_{32} \end{matrix} & \begin{matrix} t_x \\ t_y \\ t_z \end{matrix} \end{pmatrix}$$

rotation R      translation T

# The Homography Matrix

- all rotation matrices are orthonormal, i.e.  $\sqrt{r_{11}^2 + r_{21}^2 + r_{31}^2} = 1$   
 $\sqrt{r_{12}^2 + r_{22}^2 + r_{32}^2} = 1$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \begin{matrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ r_{31} & r_{32} \end{matrix} & \begin{matrix} t_x \\ t_y \\ t_z \end{matrix} \end{pmatrix} = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix}$$

$H$

let's call this  
"homography matrix"



# Understanding Pose Estimation

1. how to get projected 2D coordinates?
2. image formation
3. estimate pose with linear homography method
  - how to compute the homography matrix
4. estimate pose with nonlinear Levenberg-Marquardt method (next class)
  - how to get position and rotation from that matrix

# The Homography Matrix

Turns out that: any homography matrix has only 8 degrees of freedom – can scale matrix by  $s$  and get the same 3D-to-2D mapping

- image formation with scaled homography matrix  $sH$

$$\begin{pmatrix} x_i^n \\ y_i^n \end{pmatrix} = \begin{pmatrix} \frac{x_i^c}{w_i^c} \\ \frac{y_i^c}{w_i^c} \end{pmatrix} = \begin{pmatrix} \frac{sh_1 x_i + sh_2 y_i + sh_3}{sh_7 x_i + sh_8 y_i + sh_9} \\ \frac{sh_4 x_i + sh_5 y_i + sh_6}{sh_7 x_i + sh_8 y_i + sh_9} \end{pmatrix} = \begin{pmatrix} \frac{\cancel{s}(h_1 x_i + h_2 y_i + h_3)}{\cancel{s}(h_7 x_i + h_8 y_i + h_9)} \\ \frac{\cancel{s}(h_4 x_i + h_5 y_i + h_6)}{\cancel{s}(h_7 x_i + h_8 y_i + h_9)} \end{pmatrix}$$

# The Homography Matrix

- common approach: estimate a scaled version of the homography matrix, where  $h_9 = 1$
- we will see later how we can get scale factor  $s$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix}$$

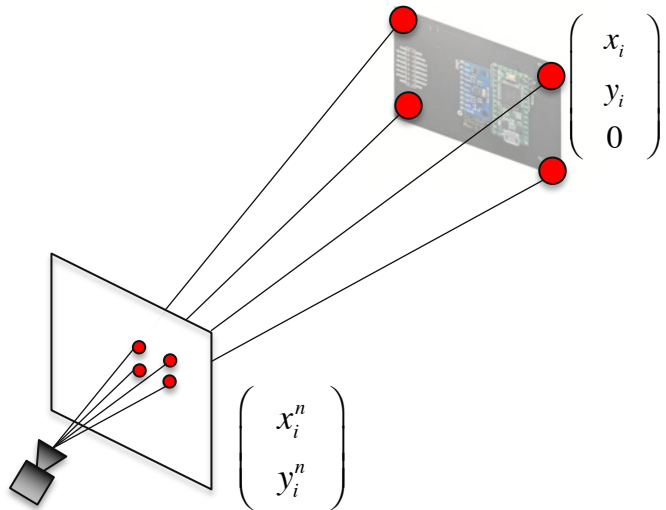
← estimate these 8 homography matrix elements!

# Pose Estimation via Homography

- image formation changes to

$$\begin{pmatrix} x_i^c \\ y_i^c \\ w_i^c \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

↑  
homography matrix with  
8 unknowns!

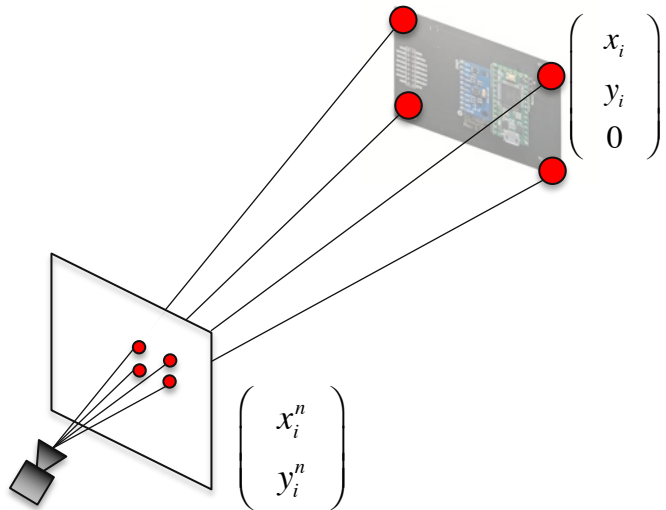


# Pose Estimation via Homography

- image formation changes to

$$\begin{pmatrix} x_i^c \\ y_i^c \\ w_i^c \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x_i^n \\ y_i^n \end{pmatrix} = \begin{pmatrix} \frac{x_i^c}{w_i^c} \\ \frac{y_i^c}{w_i^c} \end{pmatrix} = \begin{pmatrix} \frac{h_1 x_i + h_2 y_i + h_3}{h_7 x_i + h_8 y_i + 1} \\ \frac{h_4 x_i + h_5 y_i + h_6}{h_7 x_i + h_8 y_i + 1} \end{pmatrix}$$



# Pose Estimation via Homography

- multiply by denominator

$$\begin{pmatrix} x_i^n \\ y_i^n \end{pmatrix} = \begin{pmatrix} \frac{x_i^c}{w_i^c} \\ \frac{y_i^c}{w_i^c} \end{pmatrix} = \begin{pmatrix} \frac{h_1 x_i + h_2 y_i + h_3}{h_7 x_i + h_8 y_i + 1} \\ \frac{h_4 x_i + h_5 y_i + h_6}{h_7 x_i + h_8 y_i + 1} \end{pmatrix}$$



$$\begin{aligned} x_i^n (h_7 x_i + h_8 y_i + 1) &= h_1 x_i + h_2 y_i + h_3 \\ y_i^n (h_7 x_i + h_8 y_i + 1) &= h_4 x_i + h_5 y_i + h_6 \end{aligned}$$

# Pose Estimation via Homography

- reorder equations

$$h_1 x_i + h_2 y_i + h_3 - h_7 x_i x_i^n - h_8 y_i x_i^n = x_i^n$$

$$h_4 x_i + h_5 y_i + h_6 - h_7 x_i y_i^n - h_8 y_i y_i^n = y_i^n$$



$$x_i^n (h_7 x_i + h_8 y_i + 1) = h_1 x_i + h_2 y_i + h_3$$

$$y_i^n (h_7 x_i + h_8 y_i + 1) = h_4 x_i + h_5 y_i + h_6$$

# Pose Estimation via Homography

- 8 unknowns (red) but only 2 measurements (blue) per 3D-to-2D point correspondence

$$\begin{aligned} \boxed{h_1} x_i + \boxed{h_2} y_i + \boxed{h_3} - \boxed{h_7} x_i \boxed{x_i^n} - \boxed{h_8} y_i \boxed{x_i^n} &= \boxed{x_i^n} \\ \boxed{h_4} x_i + \boxed{h_5} y_i + \boxed{h_6} - \boxed{h_7} x_i \boxed{y_i^n} - \boxed{h_8} y_i \boxed{y_i^n} &= \boxed{y_i^n} \end{aligned}$$

- need at least 4 point correspondences to get to invertible system with 8 equations & 8 unknowns!
- VRduino has 4 photodiodes  $\rightarrow$  need all 4 to compute pose



# Pose Estimation via Homography

- solve  $Ah=b$  on Arduino using Matrix Math Library via **MatrixInversion** function (details in lab)

$$\begin{pmatrix}
 x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 x_1^n & -y_1 x_1^n \\
 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 y_1^n & -y_1 y_1^n \\
 x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2 x_2^n & -y_2 x_2^n \\
 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2 y_2^n & -y_2 y_2^n \\
 x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3 x_3^n & -y_3 x_3^n \\
 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3 y_3^n & -y_3 y_3^n \\
 x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4 x_4^n & -y_4 x_4^n \\
 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4 y_4^n & -y_4 y_4^n
 \end{pmatrix}
 \begin{pmatrix}
 h_1 \\
 h_2 \\
 h_3 \\
 h_4 \\
 h_5 \\
 h_6 \\
 h_7 \\
 h_8
 \end{pmatrix}
 =
 \begin{pmatrix}
 x_1^n \\
 y_1^n \\
 x_2^n \\
 y_2^n \\
 x_3^n \\
 y_3^n \\
 x_4^n \\
 y_4^n
 \end{pmatrix}$$

$A$ 
 $h$ 
 $b$

# Get Position from Homography Matrix

- still need scale factor  $s$  to get position!

just computed this

↓

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix}$$

# Get Position from Homography Matrix

- normalize homography to have approx. unit-length columns for the rotation part, such that  $\sqrt{r_{11}^2 + r_{21}^2 + r_{31}^2} \approx 1$ ,  $\sqrt{r_{12}^2 + r_{22}^2 + r_{32}^2} \approx 1$

$$s = \frac{2}{\sqrt{h_1^2 + h_4^2 + h_7^2} + \sqrt{h_2^2 + h_5^2 + h_8^2}}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix}$$

# Get Position from Homography Matrix

- this gives us the position as

$$t_x = sh_3, \quad t_y = sh_6, \quad t_z = -s$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix}$$

# Get Rotation from Homography Matrix

1. get normalized 1<sup>st</sup> column of 3x3 rotation matrix
2. get normalized 2<sup>nd</sup> column via orthogonalization
3. get missing 3<sup>rd</sup> column with cross product

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix}$$

# Get Rotation from Homography Matrix

1. get normalized 1<sup>st</sup> column of 3x3 rotation matrix

$$\tilde{r}_1 = \begin{pmatrix} h_1 \\ h_4 \\ -h_7 \end{pmatrix}, \quad r_1 = \frac{\tilde{r}_1}{\|\tilde{r}_1\|_2}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix}$$

# Get Rotation from Homography Matrix

2. get normalized 2<sup>nd</sup> column via orthogonalization

$$\tilde{r}_2 = \begin{pmatrix} h_2 \\ h_5 \\ -h_8 \end{pmatrix} - \left( r_1 \cdot \begin{pmatrix} h_2 \\ h_5 \\ -h_8 \end{pmatrix} \right) r_1, \quad r_2 = \frac{\tilde{r}_2}{\|\tilde{r}_2\|_2}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix}$$

# Get Rotation from Homography Matrix

3. get missing 3<sup>rd</sup> column with cross product:  $r_3 = r_1 \times r_2$
- $r_3$  this is guaranteed to be orthogonal to the other two columns

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix}$$



# Get Rotation from Homography Matrix

- make 3x3 rotation matrix  $R = (r_1 \ r_2 \ r_3) = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$
- convert to quaternion or Euler angles

# Get Rotation from Homography Matrix

- remember Euler angles (with yaw-pitch-roll order):

$$\underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}}_{\mathbf{R}} = \underbrace{\begin{pmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{R}_z(\theta_z)} \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{pmatrix}}_{\mathbf{R}_x(\theta_x)} \underbrace{\begin{pmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{pmatrix}}_{\mathbf{R}_y(\theta_y)}$$
$$= \begin{pmatrix} \cos(\theta_y) \cos(\theta_z) - \sin(\theta_x) \sin(\theta_y) \sin(\theta_z) & -\cos(\theta_x) \sin(\theta_z) & \sin(\theta_y) \cos(\theta_z) + \sin(\theta_x) \cos(\theta_y) \sin(\theta_z) \\ \cos(\theta_y) \sin(\theta_z) + \sin(\theta_x) \sin(\theta_y) \cos(\theta_z) & \cos(\theta_x) \cos(\theta_z) & \sin(\theta_y) \sin(\theta_z) - \sin(\theta_x) \cos(\theta_y) \cos(\theta_z) \\ -\cos(\theta_x) \sin(\theta_y) & \sin(\theta_x) & \cos(\theta_x) \cos(\theta_y) \end{pmatrix}$$

- get angles from 3x3 rotation matrix:

$$r_{32} = \sin(\theta_x)$$

$$\Rightarrow \theta_x = \sin^{-1}(r_{32}) = \text{asin}(r_{32})$$

$$\frac{r_{31}}{r_{33}} = -\frac{\cos(\theta_x) \sin(\theta_y)}{\cos(\theta_x) \cos(\theta_y)} = -\tan(\theta_y)$$

$$\Rightarrow \theta_y = \tan^{-1}\left(-\frac{r_{31}}{r_{33}}\right) = \text{atan2}(-r_{31}, r_{33})$$

$$\frac{r_{12}}{r_{22}} = -\frac{\cos(\theta_x) \sin(\theta_z)}{\cos(\theta_x) \cos(\theta_z)} = -\tan(\theta_z)$$

$$\Rightarrow \theta_z = \tan^{-1}\left(-\frac{r_{12}}{r_{22}}\right) = \text{atan2}(-r_{12}, r_{22})$$

# Temporal Filter to Smooth Noise

- pose estimation is very sensitive to noise in the measured 2D coordinates!
  - estimated position and especially rotation may be noisy
- apply a simple temporal filter with weight  $\alpha$  to smooth the pose at time step  $k$ :

$$\left(\theta_x, \theta_y, \theta_z, t_x, t_y, t_z\right)_{filtered}^{(k)} = \alpha \left(\theta_x, \theta_y, \theta_z, t_x, t_y, t_z\right)_{filtered}^{(k-1)} + (1 - \alpha) \left(\theta_x, \theta_y, \theta_z, t_x, t_y, t_z\right)_{unfiltered}^{(k)}$$

- smaller  $\alpha \rightarrow$  less filtering, larger  $\alpha \rightarrow$  more smoothing

# Pose Estimation via Homographies – Step-by-Step

in each loop() call of the VRduino:

1. get timings from all 4 photodiodes in “ticks”
2. convert “ticks” to degrees and then to 2D coordinates on plane at unit distance (i.e. get  $x_i^n, y_i^n$  )
3. populate matrix A using the 2D and 3D point coordinates
4. estimate homography as  $h=A^{-1}b$
5. get position  $t_x, t_y, t_z$  and rotation, e.g. in Euler angles from the estimated homography
6. apply temporal filter to smooth out noise

*Must read: course notes on tracking!*

# Understanding Pose Estimation

1. how to get projected 2D coordinates?
2. image formation
3. estimate pose with linear homography method
4. estimate pose with nonlinear Levenberg-Marquardt method (next class)
  - advanced topic
  - all details of this are also derived in course notes